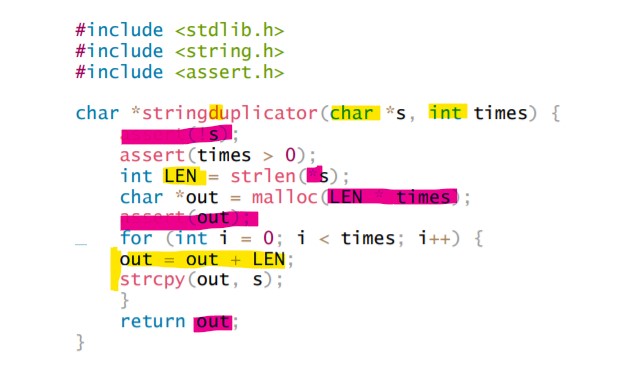
# מגישים:

|  |  |
| --- | --- |
| שי גולדנברג | שניר הורדן |
| 325382919 | 205689581 |

# 2.1.1 סעיף א

שגיאות קונבנציה מסומנות בצהוב ושגיאות תכנות מסומנות בסגול. 

קונבנציה:

1. d קטנה במקום גדולה
2. const לערכים בארגומנט של הפונקציה כי הם לא עומדים להשתנות.
3. LEN באותיות גדולות במקום בקטנות
4. = במקום =+ שמתאים יותר

שגיאות תכנות:

1. Assert(s) במקום לבדוק ששונה מNULL באמת גם לא בדיבאג.
2. S\* במקום s
3. הגודל של המאלוק לא טוב, צריך להתחשב ב+1 לתו האחרון ולהכפיל בגודל של char
4. שוב, צריך באמת לבדוק שהזיכרון הצליח גם במצב שלא דיבאג
5. Out זה סוף המחרוזת במקום תחילתה

# 2.1.2 סעיף ב

Text

Description automatically generated

# 2.2 מיזוג רשימות מקושרות ממוינות

void deleteNode( Node list )  
{  
 if( !list )  
 return;  
 deleteNode( list->next );  
 free( list );  
}

void addNodeAndPromoteList(Node list, Node \*toBeAddedAndPromoted){  
 list->x = (\*toBeAddedAndPromoted)->x;  
 \*toBeAddedAndPromoted = (\*toBeAddedAndPromoted)->next;  
}  
ErrorCode mergeSortedLists( Node list1, Node list2, Node \*mergedOut){  
 if(list1 == NULL || list2 == NULL || mergedOut == NULL){  
 return *EMPTY\_LIST*;  
 }  
 if(!isListSorted(list1) || !isListSorted(list2)){  
 return *UNSORTED\_LIST*;  
 }  
 Node start = createNode(0,NULL);  
 if(start == NULL){  
 \*mergedOut = NULL;  
 return *MEMORY\_ERROR*;  
 }  
 if(list1->x < list2->x){  
 addNodeAndPromoteList(start,&list1);  
 }  
 else{  
 assert(list2->x <= list1->x);  
 addNodeAndPromoteList(start,&list2);  
 }  
 Node run = start;  
 while (list1 != NULL || list2 != NULL){  
 run->next = createNode(0,NULL); //0 is default  
 run = run->next;  
 if(run == NULL){  
 deleteNode(start);  
 \*mergedOut = NULL;  
 return *MEMORY\_ERROR*;  
 }  
 if(list1 == NULL){  
 addNodeAndPromoteList(run,&list2);  
 }  
 else if(list2 == NULL) {  
 addNodeAndPromoteList(run,&list1);  
 }  
 else if(list1->x < list2->x){  
 addNodeAndPromoteList(run,&list1);  
 }  
 else{  
 assert(list2->x <= list1->x);  
 addNodeAndPromoteList(run,&list2);  
 }  
 }  
 (\*mergedOut)->x = start->x;  
 (\*mergedOut)->next = start->next; //where is start->next  
 return *SUCCESS*;  
}